Agenda

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
Agenda

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
Distributed Stream Processing Systems

- Stream processing systems:
  1. Centralized (traditional): manage stream data located on a single machine or closely linked machines
  2. Distributed: manage multiple parallel stream data originated from physically distributed sources (e.g. IP monitoring)

- Centralized systems use algorithms that ignore communication-efficiency issues
  -painfully inadequate for distributed applications (Sensor network monitoring)
Agenda

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
From Centralized to Distributed Stream Processing Systems

Challenges when Designing a Distributed Stream Processing System

- Related to the stream nature of data
  - Space Efficiency
  - Time Efficiency
  - Processing Efficiency

- Related to the physical distribution of data sites
  - Designing a distributed version of data processing algorithms
  - Communication cost amongst sites (communication efficiency)
  - Load balancing between sites
  - Availability in the presence of failure
From Centralized to Distributed Stream Processing Systems

Challenges when Designing a Distributed Stream Processing System

Related to the stream nature of data
- Space Efficiency
- Time Efficiency
- Processing Efficiency

Related to the physical distribution of data sites
- Designing a distributed version of data processing algorithms
- Communication cost amongst sites (communication efficiency)
- Load balancing between sites
- Availability in the presence of failure
From Centralized to Distributed Stream Processing Systems

Challenges when Designing a Distributed Stream Processing System

Related to the stream nature of data
- Space Efficiency
- Time Efficiency
- Processing Efficiency

Related to the physical distribution of data sites
- Designing a distributed version of data processing algorithms
- Communication cost amongst sites (communication efficiency)
- Load balancing between sites
- Availability in the presence of failure
From Centralized to Distributed Stream Processing Systems

Challenges when Designing a Distributed Stream Processing System

- Related to the stream nature of data:
  - Space Efficiency
  - Time Efficiency
  - Processing Efficiency

- Related to the physical distribution of data sites:
  - Designing a distributed version of data processing algorithms
  - Communication cost amongst sites (communication efficiency)
  - Load balancing between sites
  - Availability in the presence of failure
Agenda

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
Taxonomy of Distributed Stream Processing System

W.r.t. the class of queries applied over systems:
- (Non)-Holistic queries
- Duplicate- (in)sensitive queries
- Complex correlation queries

W.r.t. the communication model:
- Hierarchical
- Fully-distributed

W.r.t. the querying model:
- One-Shot DSP Systems
- Continuous DSP Systems
Taxonomy of Distributed Stream Processing System

W.r.t. the class of queries applied over systems
- (Non)-Holistic queries
- Duplicate-(in)sensitive queries
- Complex correlation queries

W.r.t. the communication model
- Hierarchical
- Fully-distributed

W.r.t. the querying model
- One-Shot DSP Systems
- Continuous DSP Systems
Taxonomy of Distributed Stream Processing System

W.r.t. the class of queries applied over systems
- (Non)-Holistic queries
- Duplicate-(in)sensitive queries
- Complex correlation queries

W.r.t. the communication model
- Hierarchical
- Fully-distributed

W.r.t. the querying model
- One-Shot DSP Systems
- Continuous DSP Systems
Taxonomy of Distributed Stream Processing System

W.r.t. the class of queries applied over systems:
- (Non)-Holistic queries
- Duplicate-(in)sensitive queries
- Complex correlation queries

W.r.t. the communication model:
- Hierarchical
- Fully-distributed

W.r.t. the querying model:
- One-Shot DSP Systems
- Continuous DSP Systems
Hierarchical vs. Fully Distributed DSP Systems

- The characteristics of underlying network communication protocol have an impact on the design of the DSP system.

- One coordinator is responsible for answering queries over the $n$ distributed streams.

- No centralized authority, the goal is having an agreement on the answer of a query.
Taxonomy of Distributed Stream Processing System

W.r.t. the class of queries applied over systems:
- (Non)-Holistic queries
- Duplicate-(in)sensitive queries
- Complex correlation queries

W.r.t. the communication model:
- Hierarchical
- Fully-distributed

W.r.t. the querying model:
- One-Shot DSP Systems
- Continuous DSP Systems
Taxonomy of Distributed Stream Processing System

W.r.t. the class of queries applied over systems
- (Non)-Holistic queries
- Duplicate-(in)sensitive queries
- Complex correlation queries

W.r.t. the communication model
- Hierarchical
- Fully-distributed

W.r.t. the querying model
- One-Shot DSP Systems
- Continuous DSP Systems
Agenda

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
Continuous DSP Systems

✧ Practical motivations for such systems

✧ Theoretical motivations for such systems

✧ The adaptive slack allocation concept in such systems
Motivating Scenario

**Application [Akyiyildiz et al. 2005]**

- Let $m$ sensor nodes be **distributed** in an underwater acoustic monitoring system
- **Task**: each node keeps track of certain school of fishes based on a given wave length and reports the results to a central base stations
- The base station maintains a $k$-clustering of the schools
- **Target**: $k$ attracting or dispelling acoustic devices can be deployed near the $k$ center points to use minimum energy to cover the whole region
Motivating Scenario

Application [Akyiyildiz et al. 2005]

- Let $m$ sensor nodes be **distributed** in an underwater acoustic monitoring system
- **Task**: each node keeps track of certain school of fishes based on a given wave length and reports the results to a central base stations
- The base station maintains a $k$-clustering of the schools
- **Target**: $k$ attracting or dispelling acoustic devices can be deployed near the $k$ center points to use minimum energy to cover the whole region

Settings

- Underwater sensor networks are a particularly resource constrained because of physical conditions (reduced channel capacity, harsh environment).
Motivating Scenario

Application [Akyiyildiz et al. 2005]

- Let $m$ sensor nodes be distributed in an underwater acoustic monitoring system
- **Task**: each node keeps track of certain school of fishes based on a given wave length and reports the results to a central base stations
- The base station maintains a $k$-clustering of the schools
- **Target**: $k$ attracting or dispelling acoustic devices can be deployed near the $k$ center points to use minimum energy to cover the whole region

Settings

- Underwater sensor networks are a particularly resource constrained because of physical conditions (reduced channel capacity, harsh environment).
- **Nodes**: unattached for a long time [or not at all] (lifetime= battery lifetime)
**K-Center Clustering**

### Offline approach
- Given a group $|P| = n$, find: $k \leq n$ centers for disks with smallest radius $R$ to cover $P$.
- Out of $\binom{n}{k}$ possible ways of choosing $k$ center from $n$ points, find the one which minimizes the cost.
- $NP$-hard to find better than 2-approximation to the optimal clustering [Feder et al., 1988].
- Approximation solutions to the optimal clustering are sought.
**K-Center Clustering**

**Offline approach**
- Given a group $|P| = n$, find: $k \leq n$ centers for disks with smallest radius $R$ to cover $P$
- Out of $\binom{n}{k}$ possible ways of choosing $k$ center from $n$ points, find the one which minimizes the cost
- $NP$-hard to find better than 2-approximation to the optimal clustering [Feder et al., 1988]
- Approximation solutions to the optimal clustering are soughted

**Online (Incremental) approach**
- Finds a current solution $S = \{c_1, c_2, \ldots, c_k, R\}$ of first stream points
- Continuously updates $S$ to keep it valid
Distributed Clustering

Motivation

- Lower power **locally** $\Rightarrow$ less energy consumption
- Lower possibility of message **loss** and **interference**

Suggested Global Clustering

- Coordinator performs another $k$-center clustering on $k \times m$ points
- Estimate the residual energy of nodes $\Rightarrow$ change coordinator
The PG Algorithm (Initialization phase on the central node)

- Suppose we know $R$. Pick an arbitrary point as the first center $C = \{c_1\}$
- For each $p \in P$ compute: $r_p = \min_{c \in C} d(p, c)$
- If $r_p > R \Rightarrow C = C \cup \{p\}$
- Since we do not know $R$ in advance, we make multiple guessing of $R$ as $(1 + \frac{\varepsilon}{2}), (1 + \frac{\varepsilon}{2})^2, (1 + \frac{\varepsilon}{2})^3, \ldots$
- Bound the number of guesses to $\Delta = \max_{p, q \in P} / \min_{p, q \in P, p \neq q} d(p, q)$
- Run the algorithm in parallel on each of these
- Store the resulted $O(\frac{k}{\varepsilon} \log \Delta)$ points
The PG Algorithm (Initialization phase on the central node)

- Suppose we know $R$. Pick an arbitrary point as the first center $C = \{c_1\}$
- For each $p \in P$ compute: $r_p = \min_{c \in C} d(p, c)$
- If $r_p > R \Rightarrow C = C \cup \{p\}$
- Since we do not know $R$ in advance, we make multiple guessing of $R$ as $(1 + \frac{\varepsilon}{2}), (1 + \frac{\varepsilon}{2})^2, (1 + \frac{\varepsilon}{2})^3, \ldots$
- Bound the number of guesses to $\Delta = \max_{p, q \in P} / \min_{p, q \in P, p \neq q} d(p, q)$
- Run the algorithm in parallel on each of these
- Store the resulted $O(\frac{k}{\varepsilon} \log \Delta)$ points

Clustering quality and storage

- $(2 + \varepsilon)$—approximation to optimal clustering is guaranteed
- Stores at most $O(\frac{k}{\varepsilon} \log \Delta)$ (do not save guesses larger than $\max_{p, q} d(p, q)$ or smaller than $\min_{p, q} d(p, q)$ or those who result in more than $k$ centers)
Global Parallel Guessing [Cromode et al., ICDE 2007]

The PG Algorithm (Running Phase)

1. While there is input stream point \( p \) compute: \( r_p = \min_{c \in C} d(p, c) \)
2. If \( r_p > R \) Then
3. If \( |C| < k \) Then
4. \( C = C \cup \{p\} \)
5. update the central node with the new center
6. else
7. ask the central node for a new guess of \( R \)
9. end while

The Communication Complexity

- Worst case: all \( m \) nodes simultaneously observe a new point \( p \) for a guess \( R \) and send it to the server
- For each guess this can happen \( k \) times, there are at most \( O(\frac{1}{\epsilon} \log \Delta) \) guesses
- The communication cost is \( O(\frac{km}{\epsilon} \log \Delta) \)
The Problem at Hand

Collecting Sensor Data
The Problem at Hand

Collecting Sensor Data

(Node 1, timestamp $t_1$): $p_1, p_2, p_3, \ldots, p_k$
(Node 2, timestamp $t_2$): $p_1, p_2, p_3, \ldots, p_k$
(Node 3, timestamp $t_3$): $p_1, p_2, p_3, \ldots, p_k$
(Node $m$, timestamp $t_m$): $p_1, p_2, p_3, \ldots, p_k$

Attribute 1
Attribute 2
Attribute 3
\ldots
Attribute $k$
The Problem at Hand

Better: group the neighbours

M. Hassani, E. Müller, P. Spaús, A. Faqolli, T. Palpanas and T. Seidl Self-Organizing Energy Aware Clustering of Nodes in Sensor Networks Using Relevant Attributes
The Problem at Hand

Select coordinators

Attribute 1
Attribute 2
Attribute 3

Attribute k
The Problem at Hand

Let cluster members send their readings locally to coordinators.
The Problem at Hand

And let coordinators forward it to the base station
The Problem at Hand

Even better: let the clustering depend on the similarity of sensed data.
The Problem at Hand

Then select the best representative of each physical cluster
Use only the readings of the representatives to update the base station of the status of the whole network.
The number of nodes sensing similar data decreases as the dimensionality of sensed data gets higher.
The ECLUN Algorithm

The Initialization Phase
The ECLUN Algorithm

The Running Phase

1. Updates to the representative only when a “considerable” deviation happens from the representative readings of relevant attributes.

2. Maintenance on two levels:
   a) Node level: dropping deviating nodes, let them select other cluster
   b) Relevant attributes: dropping not-any-more relevant attributes, let all cluster members select other clusters
Agenda

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
MapReduce

- a programming model and an associated implementation for processing and generating large datasets
- Applicable to a variety of real-world tasks
- Users specify the computation using \textit{map} and \textit{reduce} functions
- The underlying runtime system automatically:
  1. Parallelizes the computation across large-scale clusters and machines
  2. Handles machine failures
  3. Schedules inter-machines communication for efficient use of network and disks
- Easy, widely used. On Google clusters daily:
  - $10^5$ jobs executed
  - 20+ petabytes of data processed
MapReduce: Programming Model

```
def getName(line):
    return (line.split(\'t\') [1], "1")

def addCounts(hist, (name, c)):
    hist[name] = hist.get(name, default=0) + c
    return hist

input = open(\'employees.txt\', \'r\')

intermediate = map(getName, input)

result = reduce(addCounts, intermediate)
```

### Question

Q: “What is the frequency of each first name?”
MapReduce: Execution Model

User Program

Assign as mapper

Assign as reducer

Split 0
Split 1
Split 2
Split 3

Mapper

Reducer

Part 0
Part 1

Input Files

Map Phase

Intermediate Files

Output Files

Marwan Hassani: Distributed Processing of Data Streams and Large Data Sets
MapReduce: Execution Model - Data Flow

**Input Files**
- Split 0:
  - John Smith
  - David Brown
  - John Miller
- Split 1:
- Split 2:
- Split 3:

**Mapper**
- John 2; David 1

**Intermediate Files**
- Fred 1; John 1;
- Fred Taylor
- John Harris
- Jack Moore

**Reducer**
- John 2
- John 3

**Output Files**
- Part 0
- Part 1
- Part 2
- Part 3
MapReduce: Execution Model - Operations

- **Split 0** → **Mapper** → **Reducer** → **Part 0**
- **Split 1** → **Mapper** → **Reducer** → **Part 1**
- **Split 2** → **Mapper** → **Reducer** → **Part 1**
- **Split 3** → **Mapper** → **Reducer** → **Part 1**

- **Input Files** → **Map Phase** → **Intermediate Files** → **Reduce Phase** → **Output Files**

**Operations:**
- Sequential scan
- Map Phase
- Intermediate Files
- Reduce Phase
- Output Files
- All-to-all, hash partitioning
- Sort-Merge
MapReduce: Execution Model - Types

Input Files

Split 0
Split 1
Split 2
Split 3

Mapper
Mapper
Mapper
Mapper

Intermediate Files

Reducer
Reducer

Map Phase
Reduce Phase

Output Files

Part 0
Part 1

(Document_range, Document_contents)

(List (name, count))

(name, count)

(name, List (count))

(List (name, count))
MapReduce: Execution Model - Placement

- **Locality Optimization feature of MapReduce**
- **Unavoidable Rack/Network traffic**
MapReduce: Discussion (1/2)

Why MapReduce is widely used?

- Easy-to-use abstraction allows expressing the simple computations while hiding messy details like:
  1. Data partitioning, placement and replication
  2. Computation placement (and replication)
  3. Number of nodes (mappers / reducers)
  4. ...

Why MapReduce and Hadoop systems are successful in distribution?

- Not all of the tasks are distributed
- Only tasks which contain local computations that can be applied to the input in any order can be distributed
MapReduce: Discussion (2/2)

- How do different classes of algorithms fit when applying on MapReduce systems?
  1. **One iteration** algorithms (e.g. single-pass clustering, kNN classification): perfectly fit
  2. **Multiple-iteration** algorithms (KMeans, Gaussian Mixture classification): partially fit (some shared data has to be shared between iterations)
  3. Multiple-Iteration algorithms with **large shared data** between iterations (SVM): do not fit

- How about **streaming algorithms**?
Streaming Computations vs. MapReduce Computations

- Motivation: with **truly massive** data sets, even a single pass over the data using **one machine** is prohibitive
  - Streaming computations must be distributed over many machines
- As **one-iteration** algorithms, how do streaming computations differ from computations performed by MapReduce?
- Algorithms written for MapReduce or Hadoop platforms contain massive, unordered, distributed (**mud**) computations

* - J. Feldman et. al. On Distributing Symmetric Streaming Computations. In SODA’08
A Model of mud Algorithms (1/4)

- mud algorithms consist of three functions:
  1. A local function to take a single input data and output a message (applied independently in parallel)
  2. An aggregation function applied to pairs of messages in any order
  3. In some cases: a final post-processing step

- An algorithmic model for mud algorithms $m = (\Phi, \oplus, \eta)$:
  - $\Phi : \Sigma \rightarrow Q$ represents the local function which maps an input item to a message
  - $\oplus : Q \times Q \rightarrow Q$ represents the aggregator which maps two messages to a single message
  - $\eta : Q \rightarrow \Sigma$ produces the final output
A Model of *mud* Algorithms (2/4)

- For any binary tree $\mathcal{T}$ with $n$ leaves and for any permutation $\pi$ of $\{1, \ldots, n\}$, let $m_{\tau, \pi}(X)$ denote the message $q \in Q$ that results from applying $\oplus$ along the topology of $\mathcal{T}$ with the sequence $\Phi(x_1), \ldots, \Phi(x_n)$ with an arbitrary $\pi$. The overall output of the mud algorithm is then $\eta(m_{\tau, \pi}(X))$ which is a function $\Sigma^n \rightarrow \Sigma$.

- The algorithm designer needs to be sure that $\eta(m_{\tau, \pi}(X))$ is independent from the particular choice of $\mathcal{T}$ and $\pi$.
  - This is to ensure the ability of the mud algorithm to serve as an abstract model of distributed computations that are independent of the underlying implementation.
A Model of *mud* Algorithms (3/4)

- An example of a *mud* algorithm $m = (\Phi, \oplus, \eta)$ for calculating the total span of a set of integers:

  - $\Phi : \Sigma \rightarrow Q; \Phi(x) = <x, x>$
  - $\oplus : Q \times Q \rightarrow Q;$
    
    $\oplus (<a_1,b_1>, <a_2,b_2>) = <\min(a_1,a_2), \max(b_1,b_2)>$
  - $\eta : Q \rightarrow \Sigma; \eta(<a,b>) = b - a$
We say that a function \( f : \Sigma^n \rightarrow \Sigma \) is computed by a mud algorithm \( A \) if \( f(X) = A(X) \) for all \( X \in \Sigma^n \).

Let \( q \in Q \), one possible application of \( \oplus \) is:

\[
\oplus (\oplus (\ldots \oplus (\oplus (q, \Phi(x_1)), \Phi(x_2)), \ldots, \Phi(x_{k-1})), \Phi(x_k))
\]

This sequential application corresponds to the conventional streaming model.
Model of Streaming Algorithms

- A streaming algorithm is given by $s = (\sigma, \eta)$ where:
  - $\sigma : Q \times \Sigma \rightarrow Q$ is an operator applied repeatedly to the input stream
  - $\eta : Q \rightarrow \Sigma$ converts the final state to the output

- Let $s^q(X)$ denotes the state of the streaming algorithm after starting at state $q$, and operating on the sequence $X \in \Sigma^n$; $X = x_1, \ldots x_k$ exactly in that order such that:
  \[ s^q(X) = \sigma(\sigma(\ldots \sigma(q, x_1), x_2), \ldots x_{k-1}), x_k ) \]

Then: the streaming algorithm computes $\eta(s^0(X))$

- We say that a streaming algorithms computes a function $f$ if $f : \Sigma^n \rightarrow \Sigma$; $f = \eta(s^0(X))$
Streaming Computations vs. MapReduce Computations

- How do mud algorithms and streaming algorithms compare?
  - Obviously any mud algorithm can be simulated by a stream algorithm in a straightforward way.

- The question: is it possible to simulate any stream algorithm using a mud algorithm?
Streaming Computations vs. MapReduce Computations

Theorem*: For any symmetric function \( f : \Sigma^n \rightarrow \Sigma \) computed by a streaming algorithm \((\sigma, \eta)\) with:

- \( g(n) \) – space and
- \( c(n) \) – communication

there exists a mud algorithm \((\Phi, \oplus, \eta)\) with:

- \( O(g^2(n)) \) – space and
- \( O(c(n)) \) – communication

that also computes \( f \)

*- J. Feldman et. al. On Distributing Symmetric Streaming Computations. In SODA'08
Proofing the Communication Complexity:

- A symmetric function \( f : \Sigma^n \rightarrow \Sigma \) is in the class MUD if there exists a \( \text{polylog}(n) \)-communication and \( \text{polylog}(n) \)-space mud algorithm \( m = (\Phi, \oplus, \eta) \) such that for all \( X \in \Sigma^n \) and all permutations \( \pi \) of \( \tau \) we have \( \eta(m_{\pi,\pi}(X)) = f(X) \).

- Similarly: a symmetric function \( f : \Sigma^n \rightarrow \Sigma \) is in the class SS if there exists a \( \text{polylog}(n) \)-communication and \( \text{polylog}(n) \)-space streaming algorithm \( s = (\sigma, \eta) \) such that for all \( X \in \Sigma^n \) we have \( \eta(s^0(X)) = f(X) \).

- \( \text{MUD} \subseteq \text{SS} \)
Proofing the Communication Complexity:

- Consider evaluating the function $f(X)$ given two disjoint portions of the input $X = X_A \cdot X_B$ in the following models:

First)

- On-way communication model (OCM)

$$f(X_A \cdot X_B) = E(D(X_A), X_B)$$

Second)

- Simultaneous communication model SCM

$$f(X_A \cdot X_B) = C(A(X_A), B(X_B))$$
Proofing the Communication:

- $f$ can be computed in SCM as follows:

- Because $f$ is symmetric and the sequential nature of the streaming algorithm, one can show that:

\[ \eta(S^0(X'_A \cdot X'_B)) = f(X) \]

- Thus, using comparable communication complexity of streaming algorithms, we getting the same output.
• Any order-invariant function that can be computed by a streaming algorithm can also be computed by a mud algorithm with comparable space and communication complexity.

• mud algorithms are equivalent in power to symmetric streaming algorithms.

• For streaming applications on massive data sizes, where even single-pass algorithms are too much: MapReduce-like frameworks are powerful for symmetric functions.
Summary

- Introduction
- Centralized to Distributed Stream Processing (DSP) Systems
- Taxonomy of DSP Systems
- Examples on Continuous DSP Systems
- Distributed Processing of Large Data Sets: MapReduce
Thanks for your attention!

Questions?!